Swap Matching in Strings by Simulating Reactive Automata

Simone Faro

Department of Mathematics and Computer Science
Department of Linguistics and Humanities
University of Catania (Italy)

web-page: `http://www.dmi.unict.it/~faro`
email: `faro@dmi.unict.it`

Prague Stringology Conference
September 2, 2013

The Swap Matching Problem

Definition

The *string matching problem with swaps* is a well-studied variant of the classic string matching problem. It consists in finding all occurrences of a pattern P of length m in a text T of length n up to character swaps

Constraints

- Each character can be involved in at most one swap;
- Identical adjacent characters are not allowed to be swapped.

Example

```
P : agtgac
T : gtagatagccgatatggacacga
```

The Swap Matching Problem

Definition

The *string matching problem with swaps* is a well-studied variant of the classic string matching problem. It consists in finding all occurrences of a pattern P of length m in a text T of length n up to character swaps

Constraints

- Each character can be involved in at most one swap;
- Identical adjacent characters are not allowed to be swapped.

Example

P : agtgac
T : gtagatagccgatatggacacga

The Swap Matching Problem

Definition

The *string matching problem with swaps* is a well-studied variant of the classic string matching problem. It consists in finding all occurrences of a pattern P of length m in a text T of length n up to character swaps

Constraints

- Each character can be involved in at most one swap;
- Identical adjacent characters are not allowed to be swapped.

Example

```
P : agtgac
T : gtagatagccgatatggacacgra
```

The Swap Matching Problem

Previous Theoretical Results

- 1995 The problem was introduced by Muthukrishnan;
- 1997 The first nontrivial result was reported by Amir *et al.* who provided a $\mathcal{O}(nm^{\frac{1}{3}} \log m)$-time algorithm in the case of binary alphabet;
- 1998 Amir *et al.* obatined a $\mathcal{O}(m \log^2 m)$ solution on some restrictive cases;
- 2003 Amir *et al.* solved the problem in $\mathcal{O}(n \log m \log \sigma)$-time.

The Swap Matching Problem

Iliopoulos and Rahman

- 2008. The first attempt to provide an efficient solution to the swap matching problem without using the FFT technique has been presented by Iliopoulos and Rahman. They introduced a new graph-theoretic approach to model the problem and devised an efficient algorithm, based on the bit-parallelism technique, which runs in $\mathcal{O}((n+m)\log m)$-time, in the case of short patterns.

The Swap Matching Problem

Iliopoulos and Rahman

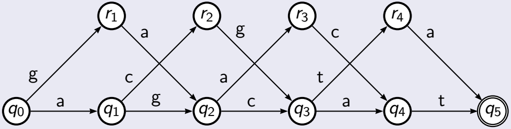
- 2008. The first attempt to provide an efficient solution to the swap matching problem without using the FFT technique has been presented by Iliopoulos and Rahman. They introduced a new graph-theoretic approach to model the problem and devised an efficient algorithm, based on the bit-parallelism technique, which runs in $\mathcal{O}((n+m)\log m)$-time, in the case of short patterns.

Cantone and Faro

- 2009. Cantone and Faro presented a new efficient algorithm, named Cross-Sampling (CS), which simulates a non-deterministic automaton with $2m$ states and $3m-2$ transitions. It admits an efficient bit-parallel implementation, named Bit-Parallel-Cross-Sampling (BPCS), which achieves $\mathcal{O}(n)$ worst-case time and $\mathcal{O}(\sigma)$ space complexity in the case of short patterns.

The Standard Automaton

Cantone & Faro (2009)

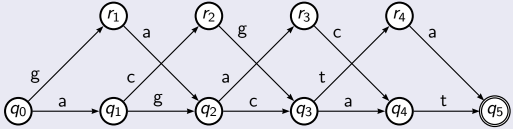


Dimension : $2m$ states and $3m - 2$ transitions

Simulation : 7 bitwise operations

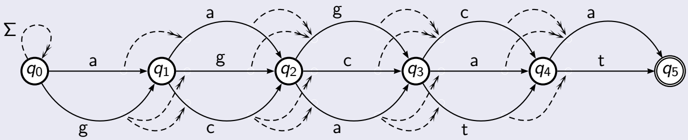
The Standard Automaton

Cantone & Faro (2009)



Dimension : $2m$ states and $3m-2$ transitions
Simulation : 7 bitwise operations

The present work (2013)



Dimension : m states, $3m-2$ transitions ($8m-12$ links)
Simulation : 7 operations (or 2 operations under suitable conditions)

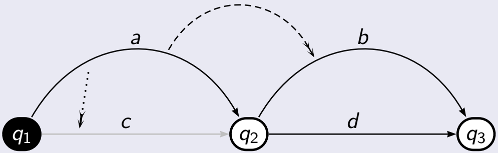
Switch Reactive Automata

Definition (Switch Reactive Transformation)

Let $\delta \subseteq (Q \times \Sigma \times Q)$ be the transition relation of an automaton A and let $\varphi \subseteq \delta$. Let T^+, T^- be two subsets of $\delta \times \delta$. A transformation $\delta \rightarrow \delta^\varphi$, for $\varphi \subseteq \delta$, is defined as follows

$$\begin{aligned}\delta^\varphi = (\delta \quad & \setminus \{\gamma \mid \gamma \in \delta \text{ and } \exists\, \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^-\}) \\ & \cup \{\gamma \mid \gamma \in \delta \text{ and } \exists\, \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^+\}\end{aligned}$$

The reactive links are intended to be applied simultaneously.



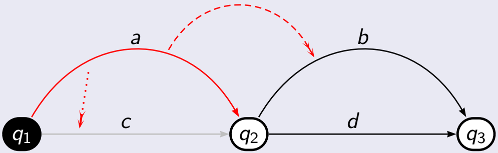
Switch Reactive Automata

Definition (Switch Reactive Transformation)

Let $\delta \subseteq (Q \times \Sigma \times Q)$ be the transition relation of an automaton A and let $\varphi \subseteq \delta$. Let T^+, T^- be two subsets of $\delta \times \delta$. A transformation $\delta \to \delta^\varphi$, for $\varphi \subseteq \delta$, is defined as follows

$$\begin{aligned}\delta^\varphi = (\delta \quad & \setminus \{\gamma \mid \gamma \in \delta \text{ and } \exists\, \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^-\}) \\ & \cup \{\gamma \mid \gamma \in \delta \text{ and } \exists\, \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^+\}\end{aligned}$$

The reactive links are intended to be applied simultaneously.



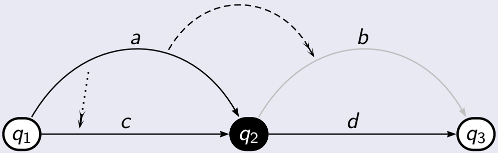
Switch Reactive Automata

Definition (Switch Reactive Transformation)

Let $\delta \subseteq (Q \times \Sigma \times Q)$ be the transition relation of an automaton A and let $\varphi \subseteq \delta$. Let T^+, T^- be two subsets of $\delta \times \delta$. A transformation $\delta \rightarrow \delta^\varphi$, for $\varphi \subseteq \delta$, is defined as follows

$$\begin{aligned}\delta^\varphi = (\delta \quad & \setminus \{\gamma \mid \gamma \in \delta \text{ and } \exists\, \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^-\}) \\ & \cup \{\gamma \mid \gamma \in \delta \text{ and } \exists\, \tau \in \varphi \text{ such that } (\tau, \gamma) \in T^+\}\end{aligned}$$

The reactive links are intended to be applied simultaneously.



Switch Reactive Automata

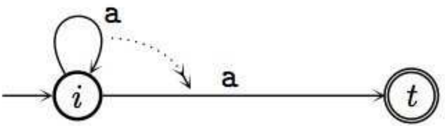
Definition (Switch Reactive Automaton)

A reactive automaton is an ordinary non-deterministic automaton with a switch reactive transformation, i.e. a triple $R = (A, T^+, T^-)$ which defines the switch reactive transformation above.

Switch Reactive Automata

Definition (Switch Reactive Automaton)

Reactive automata are used to reduce dramatically the number of states in both deterministic and the non-deterministic automata. A reactive automaton has extra links whose role is to change the behavior of the automaton itself.

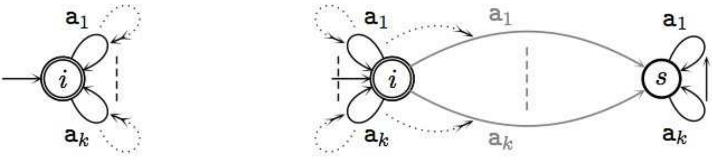


A reactive automaton accepting the language a^{2n+1}. All arcs are initially active. The automaton uses a reactive edge-to-edge link to cancel or activate the arc from the initial state i to the terminal state t.

Switch Reactive Automata

Definition (Switch Reactive Automaton)

Reactive automata are used to reduce dramatically the number of states in both deterministic and the non-deterministic automata. A reactive automaton has extra links whose role is to change the behavior of the automaton itself.

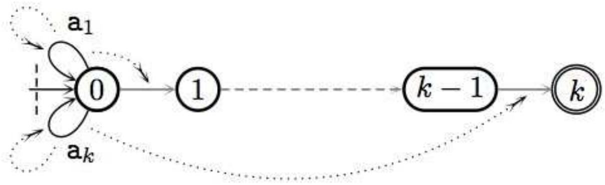


Two deterministic reactive automata accepting the set of strings in which each letter of the alphabet $\{a_1, a_2, \ldots, a_k\}$ appears at most once. All loop arcs are initially active. Loops on state i are made inactive after their first use.

Switch Reactive Automata

Definition (Switch Reactive Automaton)

Reactive automata are used to reduce dramatically the number of states in both deterministic and the non-deterministic automata. A reactive automaton has extra links whose role is to change the behavior of the automaton itself.



A deterministic reactive automaton accepting the set of strings that are permutations of the letters $a_1, a_2, \ldots, a_k$. All loops on the initial state are initially active and other ε-arcs are inactive. One reactive link for letter a_i cancels its respective loop while the second activates its associated ε-arc.

Swap Reactive Automaton

Definition (Swap Reactive Automaton)

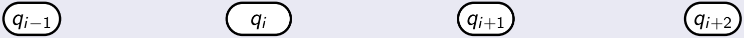
Let P be a pattern of length m over an alphabet Σ. The Swap Reactive Automaton (SRA) for P is a Reactive Automaton $R = (A, T^+, T^-)$, with $A = (Q, \Sigma, \delta, q_0, F)$, such that

- $Q = \{q_0, q_1, \ldots, q_m\}$ is the set of states;
- q_0 is the initial state;
- $F = \{q_m\}$ is the set of final states;
- δ is the transition relation;
- T^+ is the set of (switch on) reactive links;
- T^- is the set of (switch off) reactive links.

Swap Reactive Automaton

Definition (The Transition Function)

$$
\begin{array}{lll}
\delta = & \{(q_i, p_i, q_{i+1}) \mid 0 \leq i < m\} \cup & \text{no swaps} \\
 & \{(q_i, p_{i+1}, q_{i+1}) \mid 0 \leq i < m-1 \text{ and } p_i \neq p_{i+1}\} \cup & \text{start of a swap} \\
 & \{(q_i, p_{i-1}, q_{i+1}) \mid 1 \leq i < m \text{ and } p_i \neq p_{i-1}\} \cup & \text{end of a swap} \\
 & \{(q_0, \Sigma, q_0)\} & \text{self loop}
\end{array}
$$



Swap Reactive Automaton

Definition (The Transition Function)

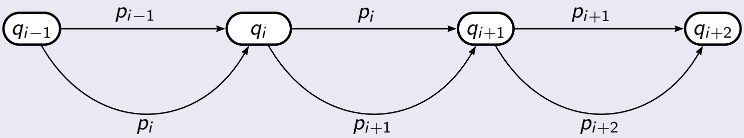
$\delta =$	$\{(q_i, p_i, q_{i+1}) \mid 0 \leq i < m\} \cup$	no swaps
	$\{(q_i, p_{i+1}, q_{i+1}) \mid 0 \leq i < m-1 \text{ and } p_i \neq p_{i+1}\} \cup$	start of a swap
	$\{(q_i, p_{i-1}, q_{i+1}) \mid 1 \leq i < m \text{ and } p_i \neq p_{i-1}\} \cup$	end of a swap
	$\{(q_0, \Sigma, q_0)\}$	self loop

$$q_{i-1} \xrightarrow{p_{i-1}} q_i \xrightarrow{p_i} q_{i+1} \xrightarrow{p_{i+1}} q_{i+2}$$

Swap Reactive Automaton

Definition (The Transition Function)

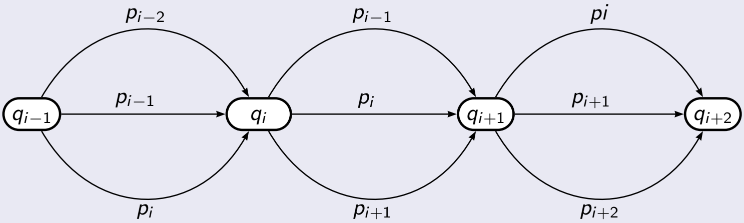
$$\delta = \begin{array}{ll} \{(q_i, p_i, q_{i+1}) \mid 0 \leq i < m\} \cup & \text{no swaps} \\ \{(q_i, p_{i+1}, q_{i+1}) \mid 0 \leq i < m-1 \text{ and } p_i \neq p_{i+1}\} \cup & \text{start of a swap} \\ \{(q_i, p_{i-1}, q_{i+1}) \mid 1 \leq i < m \text{ and } p_i \neq p_{i-1}\} \cup & \text{end of a swap} \\ \{(q_0, \Sigma, q_0)\} & \text{self loop} \end{array}$$



Swap Reactive Automaton

Definition (The Transition Function)

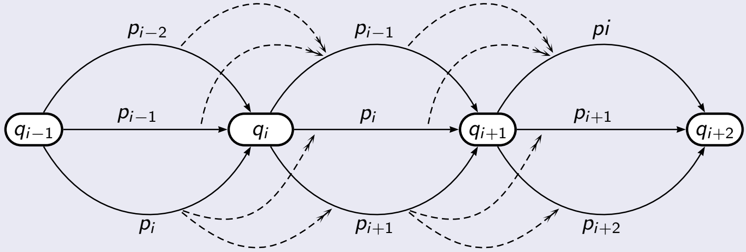
$\delta =$	$\{(q_i, p_i, q_{i+1}) \mid 0 \leq i < m\} \cup$	no swaps
	$\{(q_i, p_{i+1}, q_{i+1}) \mid 0 \leq i < m-1 \text{ and } p_i \neq p_{i+1}\} \cup$	start of a swap
	$\{(q_i, p_{i-1}, q_{i+1}) \mid 1 \leq i < m \text{ and } p_i \neq p_{i-1}\} \cup$	end of a swap
	$\{(q_0, \Sigma, q_0)\}$	self loop



Swap Reactive Automaton

Definition (Switch off reactive links)

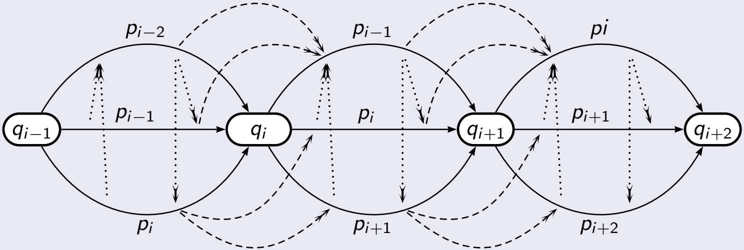
$$
\begin{aligned}
T^- = \quad & \{((q_i, p_i, q_{i+1}), (q_{i+1}, p_i, q_{i+2})) \in (\delta \times \delta) \mid 0 \le i < m-1\} \cup \\
& \{((q_i, p_{i-1}, q_{i+1}), (q_{i+1}, p_i, q_{i+2})) \in (\delta \times \delta) \mid 1 \le i < m-1\} \cup \\
& \{((q_i, p_{i+1}, q_{i+1}), (q_{i+1}, p_{i+1}, q_{i+2})) \in (\delta \times \delta) \mid 0 \le i < m-1\} \cup \\
& \{((q_i, p_{i+1}, q_{i+1}), (q_{i+1}, p_{i+2}, q_{i+2})) \in (\delta \times \delta) \mid 0 \le i < m-2\}
\end{aligned}
$$



Swap Reactive Automaton

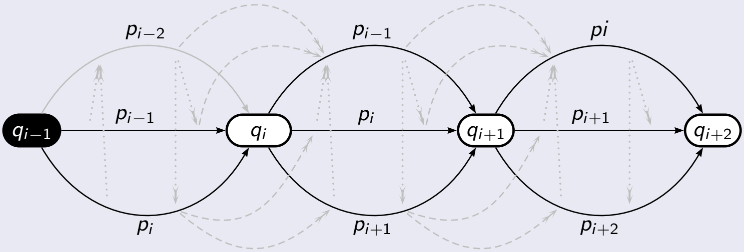
Definition (Switch on reactive links)

$$
\begin{aligned}
T^+ = \quad & \{((q_i, p_i, q_{i+1}), (q_i, p_{i-1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m-1, \} \cup \\
& \{((q_i, p_{i+1}, q_{i+1}), (q_i, p_{i-1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m-1\} \cup \\
& \{((q_i, p_{i-1}, q_{i+1}), (q_i, p_i, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m-1\} \cup \\
& \{((q_i, p_{i-1}, q_{i+1}), (q_i, p_{i+1}, q_{i+1})) \in (\delta \times \delta) \mid 0 < i < m-1\}
\end{aligned}
$$



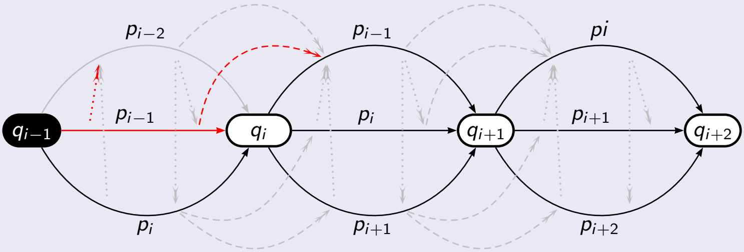
Swap Reactive Automaton

Working Principles: No Swap



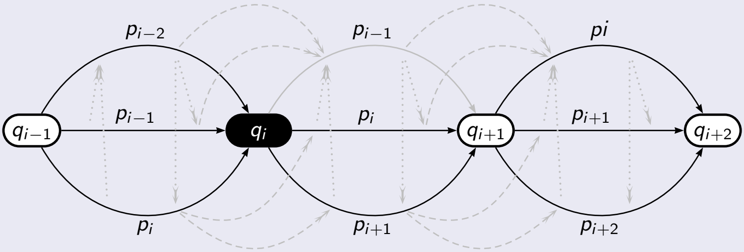
Swap Reactive Automaton

Working Principles: No Swap



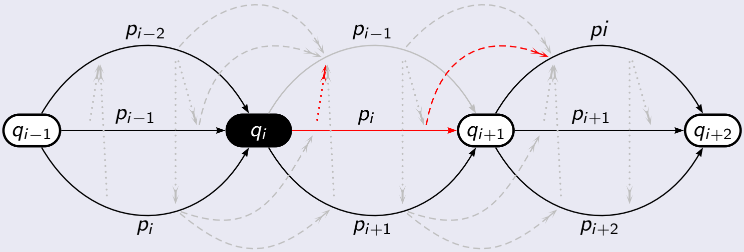
Swap Reactive Automaton

Working Principles: No Swap



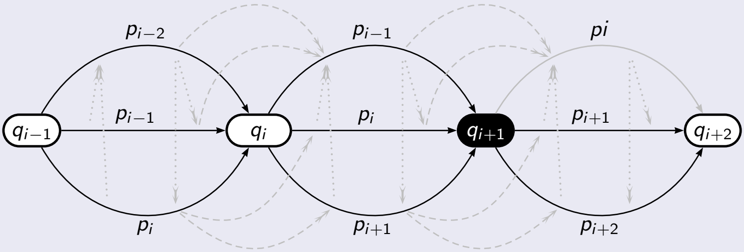
Swap Reactive Automaton

Working Principles: No Swap



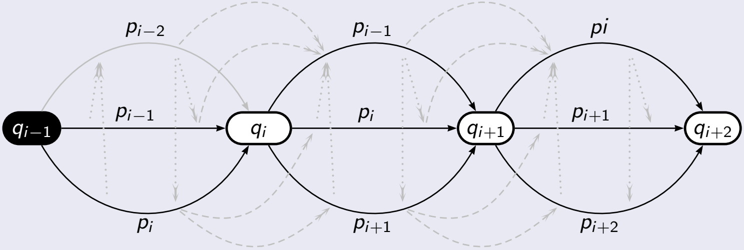
Swap Reactive Automaton

Working Principles: No Swap



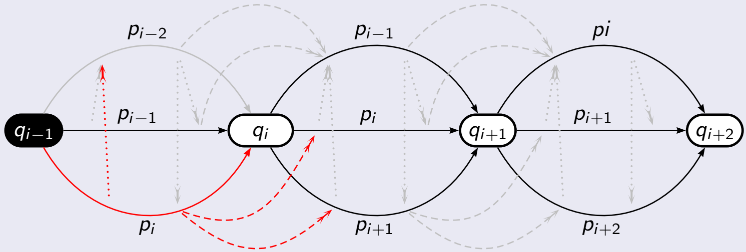
Swap Reactive Automaton

Working Principles: Swap



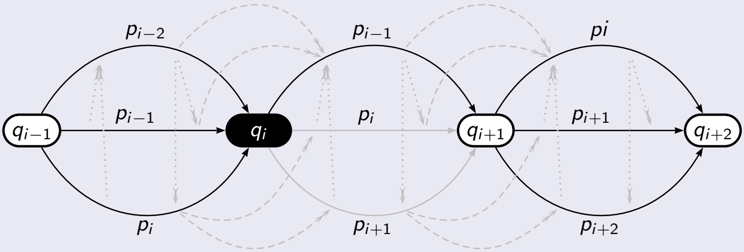
Swap Reactive Automaton

Working Principles: Swap



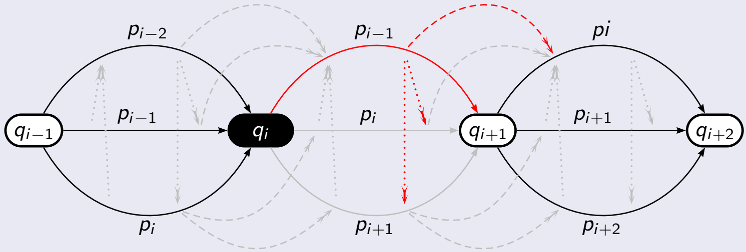
Swap Reactive Automaton

Working Principles: Swap



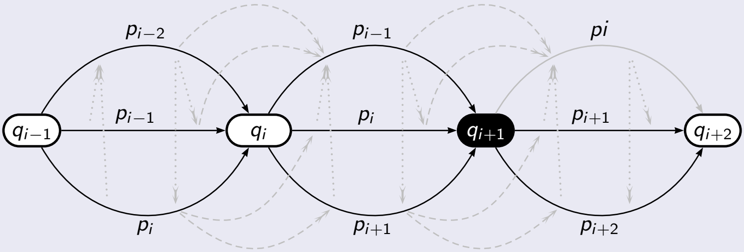
Swap Reactive Automaton

Working Principles: Swap



Swap Reactive Automaton

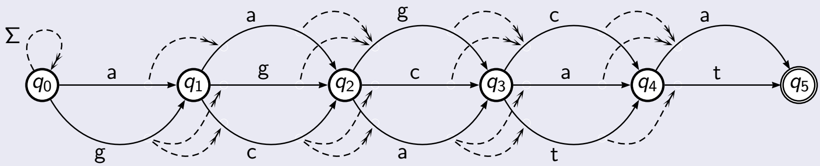
Working Principles: Swap



Swap Reactive Automaton

Example

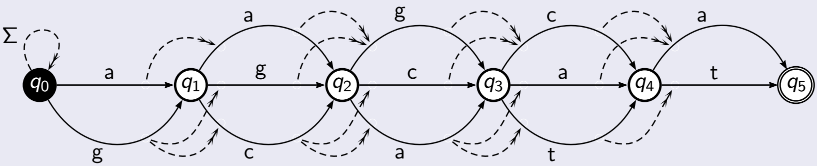
```
P : agcat
T : gaacgtagact
```



Swap Reactive Automaton

Example

```
P : agcat
T : gaacgtagact
```

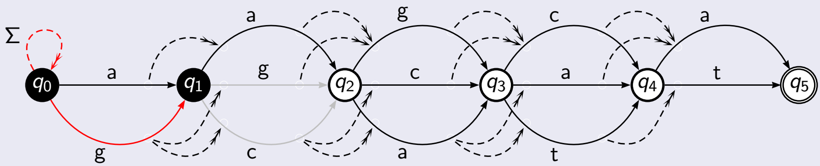


Swap Reactive Automaton

Example

P : agcat

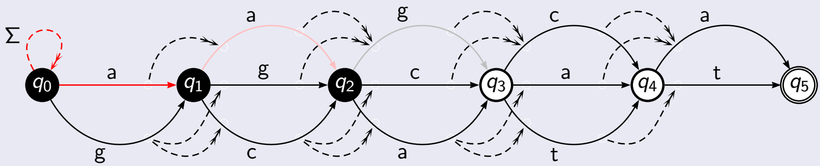
T : gaacgtagact



Swap Reactive Automaton

Example

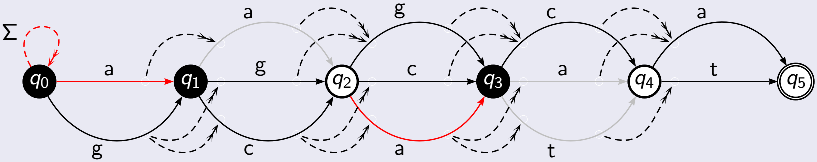
```
P : agcat
T : gaacgtagact
```



Swap Reactive Automaton

Example

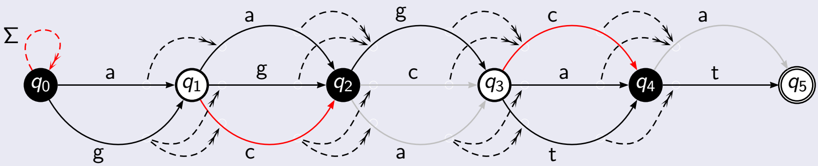
P : agcat
T : gaacgtagact



Swap Reactive Automaton

Example

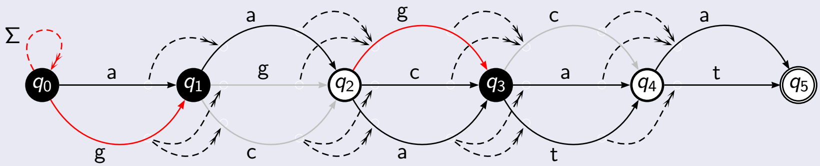
```
P : agcat
T : gaacgtagact
```



Swap Reactive Automaton

Example

P : agcat
T : gaacgtagact

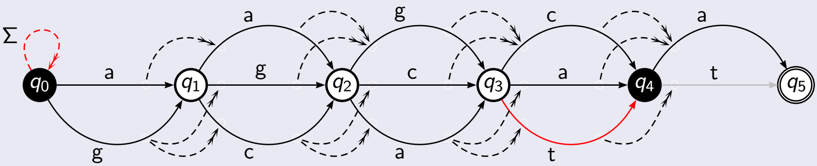


Swap Reactive Automaton

Example

P : agcat

T : gaacgtagact

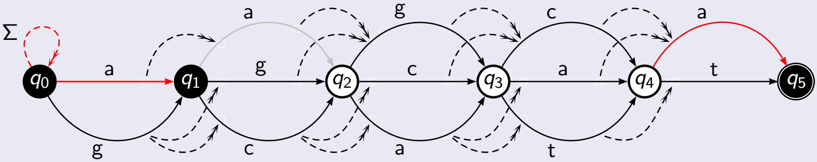


Swap Reactive Automaton

Example

P : agcat

T : gaacgtagact



The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

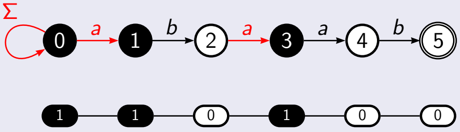
Bitwise Operations

Bitwise AND	Bitwise OR	Bitwise SHIFT
$10011010\ \&$	$10011010\ \vert$	$10011010 \gg 1$
$\underline{01011001} =$	$\underline{01011001} =$	
00011000	11011011	01001101

The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

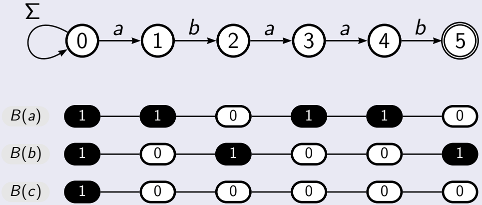
How to simulate a transition



The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

How to simulate a transition



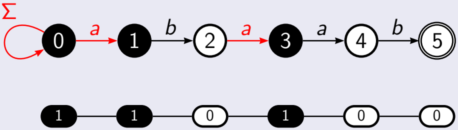
The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

How to simulate a transition

Transition on b

D

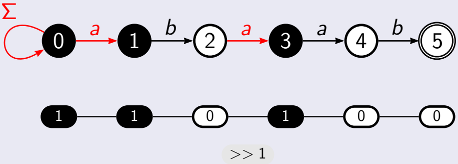


The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

How to simulate a transition

Transition on b
$D = D \gg 1$

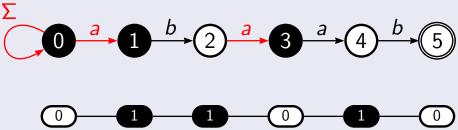


The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

How to simulate a transition

Transition on b
$D = D \gg 1$

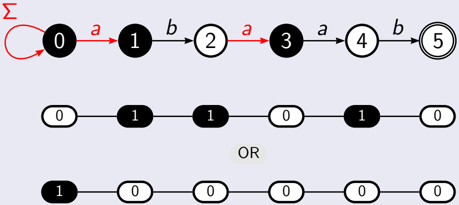


The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

How to simulate a transition

Transition on b
$D = D \gg 1$
$D = D \mid 10^{m-1}$



The Bit-Parallel Encoding

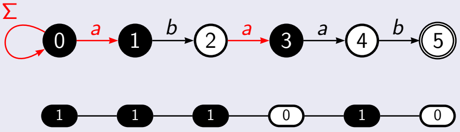
The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

How to simulate a transition

Transition on b

$D = D \gg 1$

$D = D \mid 10^{m-1}$



The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

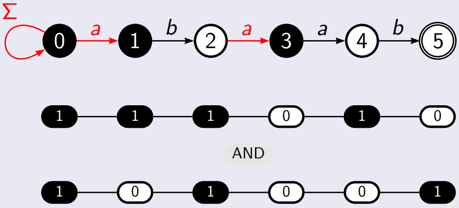
How to simulate a transition

Transition on b

$D = D \gg 1$

$D = D \mid 10^{m-1}$

$D = D \;\&\; B[b]$

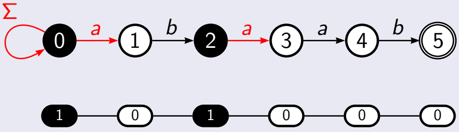


The Bit-Parallel Encoding

The bit-parallelism technique takes advantage of the intrinsic parallelism of the bitwise operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w, where w is the number of bits in the computer word.

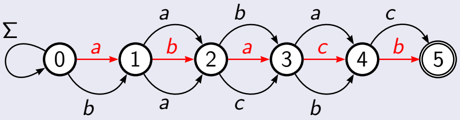
How to simulate a transition

Transition on b
$D = D \gg 1$
$D = D \mid 10^{m-1}$
$D = D \; \& \; B[b]$



The Bit-Parallel Encoding

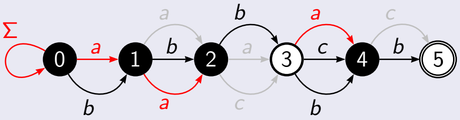
Encoding the Reactive Automaton



$B(b)$	0	1	0	0	1
$B(c)$	0	0	0	1	0
$B(a)$	1	0	1	1	0

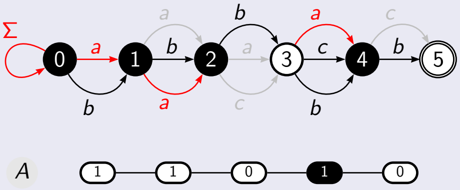
The Bit-Parallel Encoding

Encoding the Reactive Automaton



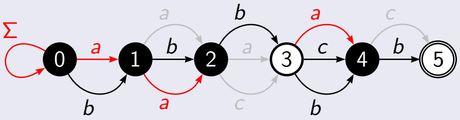
The Bit-Parallel Encoding

Encoding the Reactive Automaton



The Bit-Parallel Encoding

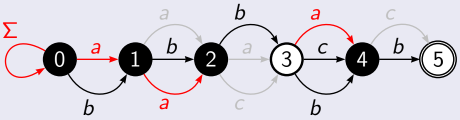
Encoding the Reactive Automaton



A	1	1	0	1	0
B	1	1	0	1	0

The Bit-Parallel Encoding

Encoding the Reactive Automaton

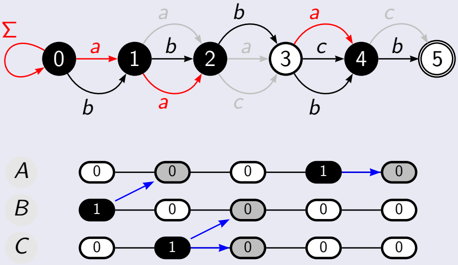


A	1	1	0	1	0
B	1	1	0	1	0
C	1	1	0	1	0

The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on *b*

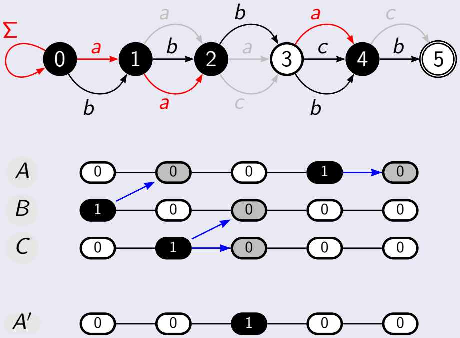


The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on b
Computing Vector A'

$A' = C \gg 1$
$A' = A' \ \& \ M[t_{j-1}]$

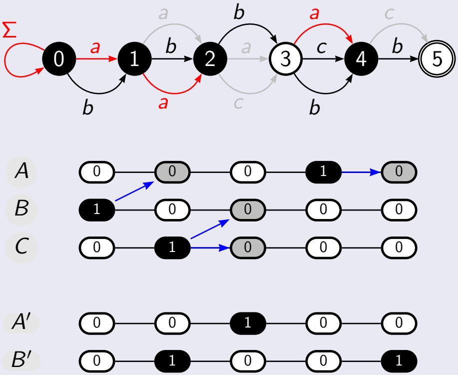


The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on b
Computing Vector B'

$B' = A \gg 1$
$B' = B' \mid (B \gg 1)$
$B' = B' \mid 10^{m-1}$
$B' = B' \ \& \ M[t_j]$

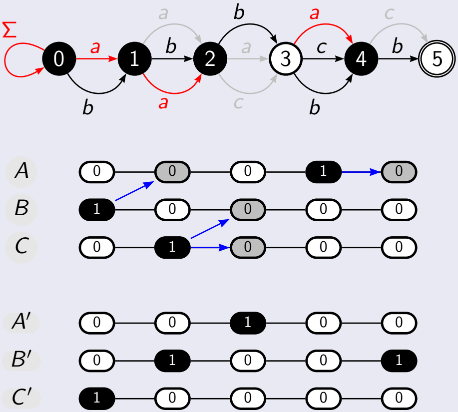


The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on b
Computing Vector C'

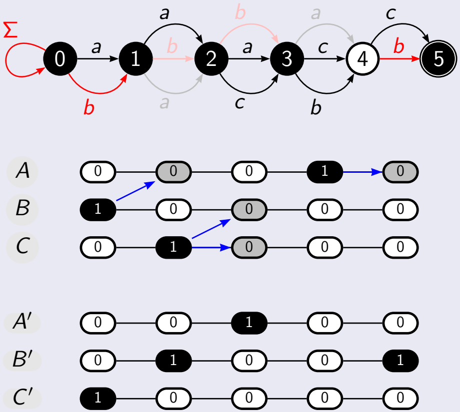
$C' = A \gg 1$
$C' = C' \mid (B \gg 1)$
$C' = C' \mid 10^{m-1}$
$C' = C' \,\&\, M[t_{j+1}]$



The Bit-Parallel Encoding

Encoding the Reactive Automaton

Transition on b



The Bit-Parallel Encoding

The Bit-Parallel Swap Reactive Automaton Matcher

BPSRA(P, m, T, n)

for $c \in \Sigma$ do
  $M[c] \leftarrow 0$
for $i \leftarrow 0$ to $m-1$ do
  $M[p_i] \leftarrow M[p_i] \mid (1 \ll i)$
$F \leftarrow 1 \ll (m-1)$
$A \leftarrow 0$
$B \leftarrow 0^{m-1}1 \ \& \ M[t_0]$
$C \leftarrow 0^{m-1}1 \ \& \ M[t_1]$
for $i \leftarrow 1$ to $n-1$ do
  $H \leftarrow (A \ll 1) \mid (M \ll 1) \mid 1$
  $A \leftarrow (C \ll 1) \ \& \ M[t_j]$
  $B \leftarrow H \ \& \ M[t_j]$
  $C \leftarrow H \ \& \ M[t_{j+1}]$
  if $((A \mid B) \ \& \ F)$ then
    output$(i - m + 1)$

A More Efficient Simulation

Definition (String With Disjoint Triplets)

A string $S = s_0 s_1 s_2 \dots s_{m-1}$, of length m, over an alphabet Σ, is a string with disjoint triplets (SDT) if $s_i \neq s_{i+2}$, for $i = 0, \dots, m-3$.

The above definition implies that in the SRA of S the standard transitions from state q_i to q_{i+1}, for $i = 0, \dots, m-1$, are labeled by different characters.

A More Efficient Simulation

Relative Frequency of SDT in different text buffers

Text	**4**	**8**	**16**	**32**
Genome Sequence	0.6080	0.2140	0.0170	0.0010
Protein Sequence	0.8420	0.6160	0.3140	0.1170
English Text	0.9380	0.8440	0.6820	0.4380
Italian Text	0.9130	0.7630	0.5100	0.2500
French Text	0.9230	0.7910	0.5930	0.3250
Chinese Text	0.9860	0.9510	0.8990	0.7750

For each text buffer data have been collected by extracting 10.000 random patterns of different length (ranging from 4 to 32) from the text, and computing the corresponding frequency of SDT.

A More Efficient Simulation

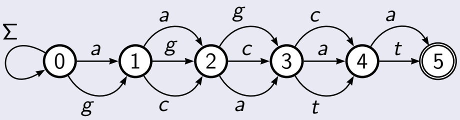
In the new proposed simulation the representation of R uses an array B of σ^2 bit-vectors, each of size m, where the i-th bit of $B[c_1, c_2]$ (which we indicate as $B[c_1, c_2]_i$) is defined as

$$B[c_1, c_2]_i = \begin{cases} 1 & \text{if } (q_i, c_1, q_{i+1}), (q_{i+1}, c_2, q_{i+2}) \in \delta \text{ and} \\ & \quad ((q_i, c_1, q_{i+1}), (q_{i+1}, c_2, q_{i+2})) \notin T^- \\ 0 & \text{otherwise} \end{cases}$$

for $c_1, c_2 \in \Sigma$, and $0 \leq i < m$.
Roughly speaking, the matrix M encodes the couples of admissible consecutive transitions in R.

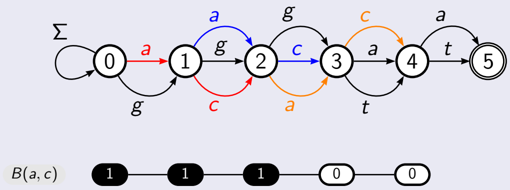
A More Efficient Simulation

How to simulate a transition



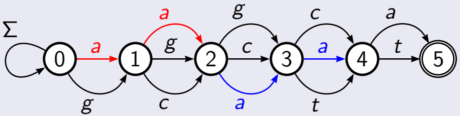
A More Efficient Simulation

How to simulate a transition



A More Efficient Simulation

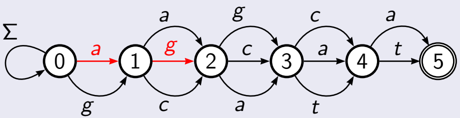
How to simulate a transition



$B(a,c)$	1	1	1	0	0
$B(a,a)$	1	0	1	0	0

A More Efficient Simulation

How to simulate a transition



$B(a,c)$	1	1	1	0	0
$B(a,a)$	1	0	1	0	0
$B(a,g)$	1	0	0	0	0

A More Efficient Simulation

- Automaton configurations are then encoded as a bit-vector D of m bits (the initial state does not need to be represented), where the i-th bit of D is set if and only if the state q_{i+1} is active.

A More Efficient Simulation

- Automaton configurations are then encoded as a bit-vector D of m bits (the initial state does not need to be represented), where the i-th bit of D is set if and only if the state q_{i+1} is active.
- When a search starts, the configuration D is initialized to $B[t_0, t_1]$. Then, while the string T is read from left to right, the automaton configuration is updated accordingly for each text character.

A More Efficient Simulation

- Automaton configurations are then encoded as a bit-vector D of m bits (the initial state does not need to be represented), where the i-th bit of D is set if and only if the state q_{i+1} is active.
- When a search starts, the configuration D is initialized to $B[t_0, t_1]$. Then, while the string T is read from left to right, the automaton configuration is updated accordingly for each text character.
- Suppose the last transition has been performed on character t_{j-1}, with $0 < j < n-1$, leading to a configuration vector D of the SRA. Then a transition on character t_j can be implemented by the bitwise operations

$$D^{(j)} = \begin{cases} B[t_0, t_1] & \text{if } j = 1 \\ (D^{(j-1)} \ll 1) \text{ and } B[t_{j-1}, t_j] & \text{otherwise} \end{cases}$$

A More Efficient Simulation

- Automaton configurations are then encoded as a bit-vector D of m bits (the initial state does not need to be represented), where the i-th bit of D is set if and only if the state q_{i+1} is active.
- When a search starts, the configuration D is initialized to $B[t_0, t_1]$. Then, while the string T is read from left to right, the automaton configuration is updated accordingly for each text character.
- Suppose the last transition has been performed on character t_{j-1}, with $0 < j < n-1$, leading to a configuration vector D of the SRA. Then a transition on character t_j can be implemented by the bitwise operations

$$D^{(j)} = \begin{cases} B[t_0, t_1] & \text{if } j = 1 \\ (D^{(j-1)} \ll 1) \text{ and } B[t_{j-1}, t_j] & \text{otherwise} \end{cases}$$

- It turns out that, if P is a SDT, then the simulation of the SRA described above works properly, as stated by the following lemma.

The Bit-Parallel Encoding

The Bit-Parallel Swap Reactive Oracle Matcher

$\mathrm{BPSRO}(P, m, T, n)$

for $c_1, c_2 \in \Sigma$ do $B[c_1, c_2] \leftarrow 0$
for $i = 1$ to $m - 1$ do
    $B[p_{i-1}, p_i] \leftarrow B[p_{i-1}, p_i] \mid (1 \ll i)$
    $B[p_i, p_{i-1}] \leftarrow B[p_i, p_{i-1}] \mid (1 \ll i)$
    if $(i < m - 1)$ then
        $B[p_{i-1}, p_{i+1}] \leftarrow B[p_{i-1}, p_{i+1}] \mid (1 \ll i)$
    if $(i > 1)$ then
        $B[p_{i-2}, p_i] \leftarrow B[p_{i-2}, p_i] \mid (1 \ll i)$
        if $(i < m - 1)$ then
            $B[p_{i-2}, p_{i+1}] \leftarrow B[p_{i-2}, p_{i+1}] \mid (1 \ll i)$
$F \leftarrow 1 \ll (m - 1)$, $D \leftarrow 0$
for $i \leftarrow 1$ to $n - 1$ do
    $D \leftarrow ((D \ll 1) \mid 1) \;\&\; B[t_{i-1}, t_i]$
    if $(D \;\&\; F)$ then
        if (P is a SDT) then output$(i - m + 1)$
        else check occurrence at position $(i - m + 1)$

Experimental results

Experimental Results

m	(A) genome sequence 2	4	8	16	32
BPCS	16.0	15.9	15.9	16.0	15.9
BPSRA	**15.4**	15.3	15.3	15.2	15.2
BPSRO	20.4	**13.7**	**11.4**	**11.2**	**11.2**

m	(B) protein sequence 2	4	8	16	32
BPCS	15.9	15.9	16.1	16.1	16.2
BPSRA	15.3	15.8	15.4	15.3	15.3
BPSRO	**12.0**	**11.2**	**11.4**	**11.3**	**11.3**

m	(C) natural language text 2	4	8	16	32
BPCS	16.0	15.9	16.1	16.3	16.0
BPSRA	15.3	15.4	15.3	15.3	15.3
BPSRO	**12.8**	**11.5**	**11.5**	**11.3**	**11.3**